VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

MATLAB-GUI pro minimalizaci logických funkcí metodou McCluskey

MATLAB-GUI for Minimising of Logical Functions Using the McCluskey Method

2012 Jiří Hrbáček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jiří Hrbáček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: MATLAB-GUI pro minimalizaci logických funkcí metodou McCluskey
MATLAB-GUI for Minimising of Logical Functions Using the McCluskey Method

Zásady pro vypracování:

Cílem práce je vytvoření grafického uživatelského prostředí (GUI) pro minimalizaci logických funkcí metodou McCluskey v systému MATLAB.

1. Popis algoritmu minimalizace logických funkcí metodou McCluskey.
2. Návrh a realizace (implementace) grafického uživatelského rozhraní (GUI) pro minimalizaci logických funkcí metodou McCluskey.
3. Vytvoření návodu pro práci s GUI.
4. Export GUI pomocí MATLAB JAVA Builderu pro MATLAB Web Server.

Seznam doporučené odborné literatury:

[1] DIVIŠ, Zdeněk; CHMELÍKOVÁ, Zdeňka; ZDRÁLEK, Jaroslav. *Logické obvody*. 2. vyd. Ostrava: VŠB - Technická univerzita Ostrava, 2008. 152 s. 978-80-248-1724-8.

[2] BERNARD, Jean Michel; HUGON, Jean; Le CORVEC, Robert. *Od logických obvodů k mikroprocesorům*. 2. nezm. vyd. Praha: SNTL - Nakladatelství technické literatury, 1988. 686 s.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Skapa, Ph.D.**

Datum zadání: 19.11.2010

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2012 .....

Rád bych na tomto místě poděkoval vedoucímu bakalářské práce Ing. Janu Skapovi, Ph.D. za věcné rady a přípomínky při tvorbě této bakalářské práce.

Abstrakt

Cílem této bakalařské práce je vytvořit aplikaci, která předvede uživateli, jak probíhá minimalizace logických funkcí McCluskeyeho metodou v jednotlivých krocích za sebou. Aplikace využívá grafického uživatelského rozhraní MATLABU a umožňuje uživateli si vyzkoušet jednotlivé kroky minimalizace, a tak lépe pochopit tuto metodu minimalizace logických funkcí.

Klíčová slova: MATLAB, GUI, logická funkce, metoda McCluskey

Abstract

The main point of this bachelor thesis is make an application, which demonstrate to user, how works minimising of logical functions by McCluskey method in single steps. Application is based on MATLAB graphic user interface and allow to try user steps of minimising to better understand this method of minimising logicals functions.

Keywords: MATLAB, GUI, logical function, McCluskey method

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CGI	–	Common Gateway Interface
CSS	–	Cascading Style Sheets
GUI	–	Graphic User Interface
HTML	–	HyperText Markup Language
JVM	–	Java Virtual Machine
MATLAB	–	MATrix LABoratory
MCR	–	MATLAB Compiler Runtime
URL	–	Uniform Resource Locator

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

S logickou funkcí se v životě setkal kdokoli z nás. V současnosti se skoro na každém kroku setkáváme s přístroji, jejíž základem je číslicový neboli digitální systém. Tento systém využívá logických obvodů ke zpracování vstupních proměnných, kdy pomocí logických operací nebo funkcí vytváří výstupní proměnné. Jelikož v dnešní době mají tyto systémy a logické funkce velký počet vstupních proměnných, je nutné z důvodů šetření nákladů nebo jednodušší realizace logických obvodů minimalizovat logické funkce do přívětivějšího tvaru. Jednou z mnoha metod minimalizace je i McCluskeyeho metoda, která je hlavním obsahem této práce.

Text je rozdělen do několika kapitol, hned v následující kapitole je stručně popsána logická funkce, rozdělení logických funkcí podle různých vlastností a jakým způsobem lze zapsat logické funkce. V další kapitole se pojednává o metodě McCluskey a postupu, který se používá k minimalizaci zadané funkce.

Čtvrtá část textu se už věnuje stěžejní části práce a to popisu grafického uživatelského rozhraní aplikace napasaného v MATLABu a implementace některých algoritmů použitých v aplikaci pro minimlaizaci logické funkce. V následující kapitole je popsán postup při nasazení aplikace na webové stránky, věnuje se MATLAB Builderu JA a komunikaci mezi appletem a servletem. V závěrečné kapitole je sepsán stručný návod pro uživatele, jak má s aplikací zacházet.

2 Logická funkce

Funkce, která navrací pro vstupní proměnné hodnoty logická 0 nebo logická 1 se označuje jako logická (někdy též nazývaná booleovská, případně dvouhodnotová) funkce.

2.1 Zadání logických funkcí

Logické funkce lze zadat ve třech tvarech, a to buď jako úplně zadanou funkci, neúplně zadanou funkci nebo rozšířenou booleovskou funkci.

Úplně zadanou booleovskou funkcí rozumíme takovou funkci, u které známe hodnoty všech možných kombinací vstupních proměnných. Celkový počet možných kombinací pro n proměnných je 2^n. Oproti tomu u neúplně zadané booleovské funkce je definována hodnota funkce jen u některých hodnot vstupních proměnných.

Pro rozšířenou booleovskou funkcí platí, že u některých hodnot kombinací vstupních proměnných není určena logická hodnota (nevíme, zda nabývá logické hodnoty 0 nebo I). Hodnotu neurčité funkce poté označujeme symbolem X.

Tabulka 1: Pravdivostní tabulka úplně zadané booleovské funkce

K	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

Tabulka 2: Pravdivostní tabulka rozšířené booleovské funkce

K	x_1	x_0	f
0	0	0	0
1	0	1	X
2	1	0	1
3	1	1	X

Tabulka 3: Pravdivostní tabulka neúplně zadané booleovské funkce

K	x_2	x_1	x_0	f
0	0	0	0	0
1	0	0	1	1
4	1	0	0	1
5	1	0	1	0

2.2 Zápisy logické funkce

Pro vyjádření logické funkce se používá několik způsobů zápisu. Mezi ty hlavní patří:

- zápis pomocí pravdivostní tabulky,

- zápis pomocí vektoru,
- číselný zápis,
- geometrický zápis,
- zápis pomocí výrazu.[1]

U vyjádření funkce pravdivostní tabulkou se všechny kombinace hodnot vstupních proměnných a jejich hodnot zapíší to tabulky, která má pro n vstupních proměnných 2^n řádků. Ukázka zápisu v tabulce 4.

Tabulka 4: Pravdivostní tabulka tří proměnných

K	x_2	x_1	x_0	f
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	I
4	1	0	0	0
5	1	0	1	I
6	1	1	0	I
7	1	1	1	I

Vektorový zápis popisuje hodnoty funkce pro vstupní proměnné v pořadí podle váhy proměnné uvedené v závorce. Například pro funkci zadanou v tabulce 4, vypadá vektorový zápis takto: $f(x_2, x_1, x_0) = 11101000$.

Číselný zápis funkce umožňuje, tak jako vektorový zápis, logické hodnoty nahradit číselnými hodnotami (logická 0 se nahradí číslem 0 a logická I se nahradí číslem 1). Díky tomu, lze vstupní proměnné vyjádřit ve dvojkové soustavě. Číselný zápis může mít dvě podoby - disjunktivní číselný zápis, kde se zapisují hodnoty vstupních vektorů, které nabývají logické hodnoty 1 a konjuktivní číselný zápis, který vyjadřuje hodnoty vstupních vektorů s hodnotou logická 0. Pro funkci v tabulce 4 má konjunktivní zápis tvar $f(x_2, x_1, x_0) = K(0, 1, 2, 4)$ a disjunktivní zápis $f(x_2, x_1, x_0) = D(3, 5, 6, 7)$.

U geometrických zápisů se většinou využívá takzvaných map, které tvoří systém "čtverečků", ve kterých jsou zapsány hodnoty funkcí jednotlivých vstupních proměnných. Nejčastěji se využívá Karnaughova mapa.

Pro zápis pomocí výrazu se využívají dva tvary zápisu. Prvním je základní součtový tvar neboli úplná disjunktivní normální forma a druhým je základní součinový tvar, často nazývány jako úplná konjunktivní normální forma. Základní součtový tvar vyjadřuje funkci jako součet základních součinů vstupních proměnných, kde součiny jsou kombinace, pro kterou funkce nabývá logické hodnoty I. Oproti tomu základní součinový tvar popisuje funkci jako součin základních součtů vstupních proměnných, kde jednotlivé součty vyjadřují kombinace, kde funkce má logickou hodnotu 0.

Základní součtový tvar funkce zadané pravdivostní tabulkou 4 je $f = \overline{x_2}x_1x_0 + x_2\overline{x_1}x_0 + x_2x_1\overline{x_0} + x_2x_1x_0$ a základní součinový tvar funkce je $f = (x_2 + x_1 + x_0)(x_2 + x_1 + \overline{x_0})(x_2 + \overline{x_1} + x_0)(\overline{x_2} + x_1 + x_0)$.

2.3 Zjednodušování zápisu logické funkce

Zjednodušování zápisu logické funkce slouží k nalezení nejjednoduššího zápisu funkce, který povede k přijatelnější technické realizaci. Funkce zadaná v úplném tvaru obsahuje často stejné proměnné a operace, které se dají zredukovat. Ve výsledku tak dojde ke snížení počtu obvodových prvků a zpřehlednění funkce. Tento proces zjednodušování se nazývá minimalizace logické funkce.

Pro minimalizace se nejčastěji používají tyto metody:

- algebraická metoda,
- minimalizace Karnaughovou metodou,
- minimalizace ostatními metodami.

U algebraické metody se využívá k minimalizaci logické funkce Booleovské algebry a DeMorganových zákonů, při minimalizaci Karnaughovou metodou se vychází z takzvaných Karnaughových map. Mezi nejčastěji využívané ostatní metody minimalizace patří McCluskeyeho metoda a Patrickova metoda [2].

Zjednodušování logických funkcí má velký význam a je často přínosem, ale neznamená to, že funkce v minimálním tvaru vede k nejlepší realizaci logického obvodu. Existují i případy, kdy z důvodů vlastností některých použitých logických členů je vhodnější pro realizaci využít jen částečně zjednodušené funkce.

3 Metoda McCluskey

Tato metoda slouží k nalezení minimalního normálního tvaru logické funkce. Funkcionalita je podobná minimalizaci pomocí Karnaughových map, ale je vhodnější pro větší počet vstupních proměnných.

Princip metody spočívá ve vzájemném porovnávaní vstupních vektorů. Nejprve dojde k roztřídění vstupních vektorů do skupin podle počtu logických hodnot. Poté dochází k porovnání mezi vektory jednotlivých skupin, které se liší svým počtem o 1. Pokud se dané porovnávané vektory liší jen v jedné proměnné, můžeme konstatovat, že vektory nezávisí na lišící se proměnné a lze je tedy sloučit do jednoho. Tak se pokračuje neustále až do stavu, kdy se necházejí v jednotlivých částech tabulky vektory, které nesplňují podmínky spojení a nelze je tedy zjednodušit - nazýváme je implikanty. Z implikantů se poté vytvoří takzvaná tabulka pokrytí, ze které určujeme minimalní normální tvar logické funkce.

3.1 Příklad řešení minimalizace logické funkce pomocí metody McCluskey

V této části je uvedeno, jak se řeší minimalizace logické funkce v součtovém tvaru metodou McCluskey. Pro ukázku byla zvolena tato rozšířená funkce:

$F(x_3, x_2, x_1, x_0)$ =D(4,8,10,11,12,15(9,14))

Tabulka 5: Pravdivostní tabulka kombinačního obvodu

K	x_3	x_2	x_1	x_0	f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	I
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	I
9	1	0	0	1	X
10	1	0	1	0	I
11	1	0	1	1	I
12	1	1	0	0	I
13	1	1	0	1	0
14	1	1	1	0	X
15	1	1	1	1	I

Funkce F je zadaná disjunktně a v součtovém tvaru, tedy do pravdivostní tabulky se zapisují vektory, které nabývají logické hodnoty I. V tomto případě se jedná o vektory $4, 8, 10, 11, 12, 15$. U vektorů 9 a 14 není funkce definována a nemusí se zahrnovat do řešení - přiřadí se jim logická hodnota 0. Jelikož by s nimi pak nemohlo být počítano při zjednodušení a mohlo by dojít k takzanému hazardnímu stavu, zahrneme je tedy do tabulky a jejich případné využití při řešení použijeme v tabulce pokrytí (při zjednodušování s nimi počítáme jako by nabývaly logické hodnoty I).

Vycházíme z tabulky 5 a vyřadíme z ní ty vektory, jejichž logická hodnota je rovna 0 a získáme tím tabulku 6, která je pravdivostní tabulkou pro zadanou funkci F.

Tabulka 6: Pravdivostní tabulka funkce F v součtovém tvaru

K	x_3	x_2	x_1	x_0
4	0	1	0	0
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
14	1	1	1	0
15	1	1	1	1

Dalším krokem je porovnání vektorů z tabulky 6 a jejich přiřazení do jednotlivých skupin, které se liší v jedné proměnné. Vektory rozdělíme podle počtu logických hodnot 1 ve vektoru a vznikne tím tabulka 7.

Tabulka 7: První etapa porovnání

K	x_3	x_2	x_1	x_0	Počet log I
4	0	1	0	0	1
8	1	0	0	0	1
9	1	0	0	1	2
10	1	0	1	0	2
12	1	1	0	0	2
11	1	0	1	1	3
14	1	1	1	0	3
15	1	1	1	1	4

V dalším kroku vycházíme z tabulky 7, kde už máme vektory rozdělené do skupin. Z tohoto rozdělení vycházíme, jelikož můžeme porovnávat pouze vektory ze sousedních skupin. V tomto příkladě se tedy porovná vektor 4 s vektory $9, 10, 12$, vektor 8 s vektory $9, 10, 12$, vektor 9 s vektory $11, 14$ a tak dále až do posledního porovnání vektorů 14 a 15. Pokud vektory splňují podmínku sloučení (liší se jen v jedné proměnné), tak je můžeme spojit. V další tabulce tedy do slouce K zapíšeme ty vektroy, které spojujeme a pozici, kde se liší nahradíme znakem „ - “. Pokud se v tabulce během porvnávání nachází vektor, který nelze s jiným sloučit (nazýváme jej implikant), tak ho zapíšeme do tabulky prvků.

Při porovnávání může docházet k tomu, že se nám vyskytnout po spojení vektory, které jsou si rovny. Můžeme je do tabulky přidat, ale není to nutné, protože stejně pak dojde k odstranění duplicity. Pro příklad z tabulky 8 po spojení vzniknou dva shodné vektory $8, 9, 10, 11$ a $8, 10, 9, 11$. Pak do tabulky stačí přidat vektor $8, 9, 10, 11$. Dále se nám v tabulce vyskytuje vektor $4, 12$, který se nepoužil během spojování. Jedná se o implikant, který označíme písmenem A a přidáme do tabulky pokrytí.

Tabulka 8: Vzájemné porovnání vektorů ve skupinách

K	x_3	x_2	x_1	x_0	Počet log I	Implikant
4,12	-	1	0	0	1	A
8,9	1	0	0	-	1	
8,10	1	0	-	0	1	
8,12	1	-	0	0	1	
9,11	1	0	-	1	2	
10,11	1	0	1	-	2	
10,14	1	-	1	0	2	
12,14	1	1	-	0	2	
11,15	1	-	1	1	3	
14,15	1	1	1	-	3	

Po vzájemném porovnání vektorů v tabulce 8 nám vznikla tabulka 9. V této tabulce už není možné spojovat vektory mezi sebou, tedy ze všech tří zbylých vektorů se staly implikanty - označíme je písmeny B, C, D a zapíšeme do tabulky pokrytí. Tímto krokem

došlo k nalezení všech implikantů a je nutné sestavit minimální normální tvar fuknce z tabulky pokrytí.

Tabulka 9: Vzájemné porovnání vektorů ve skupinách

K	x_3	x_2	x_1	x_0	Počet log I	Implikant
8,9,10,11	1	0	-	-	1	B
8,10,12,14	1	-	-	0	1	C
10,11,14,15	1	-	1	-	2	D

Tabulka 10: Tabulka pokrytí

F=1,X	4	8	10	11	12	15	9	14
A	X				X			
B		X	X	X			X	
C		X	X		X			X
D			X	X		X		X

Tabulka pokrytí obsahuje vstupní vektory, které nabývají logických hodnot I a X. Tyto vektory tvoří sloupce dané tabulky, přičemž prvně zapisujeme hodnoty těch vektorů, nabývající logický hodnot I a poté X. Volí se to takto, jelikož na minimální normální tvar funkce mají hlavní vliv ty vstupní vektory, které nabývají logické hodnoty I. Hodnoty X naleznou uplatnění tehdy, pokud se objeví v tabulce pokrytí implikanty, které pokrývají stejné vstupní vektory v první části tabulky. Pak se volí ten implikant, který pokrývá v tabulce celkově více indexů. Řádky tabulky pokrytí tvoří získáné implikanty během vzájemného porvnávání vektorů - písmenem X se poté v daném řádku a sloupci vyznačí vstupní vektory, ze kterých se implikant skládá.

V tabulce pokrytí se nacházejí implikanty, které pokrývají jako jediné určitý vstupní vektor - impikant A pokrývá vektor 4 a zároveň vektor 12, implikant D pokrývá vektor 15 a zároveň vektory 10 a 11. Zbývá nám pokrýt vektor 8, na který můžeme použít oba zbylé implikanty B a C. Jelikož oba tyto implikanty pokrývají v celé tabulce stejný počet vektorů, tak je jedno který zvolíme, protože obě řešení jsou ekvivalentní.

Výsledný minimální normální tvar funkce má tedy dvě možná řešení:

$$^1F = A + D + B = x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_3 \cdot x_1 + x_3 \cdot \overline{x_2}$$

$$^2F = A + D + C = x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_3 \cdot x_1 + x_3 \cdot \overline{x_0}$$

4 Grafické uživatelské prostředí v MATLAB-u

V programovém prostředí MATLAB se dá grafické uživatelské rozhraní aplikací vytvářet dvěma základními způsoby - programovaním přímo v takzvaných m-file (soubory s příponou *.m obsahující kód napsaný v jazyce MATLAB) nebo použitím GUIDE (grafický editor pro tvorbu GUI)[3].

Přímé vytváření GUI v souborech je náročnější než při použití GUIDE. Při této přímé tvorbě musíme vkládat jednotlivé prvky GUI přesně podle určitého systému pozicovaní (například souřadnic), ale nevýhodou je, že rozvržení objektů nevidíme a musíme proto daný m-file pokaždé spustit, abychom viděli výsledné GUI.

Při použití grafického editoru GUIDE máme možnost okamžitě vidět, jak dané GUI vypadá. Umístění jednotlivých prvků (tlačítek, textových polí, atd.) se provadí jejich přesunutím na výslednou plochu. U daných prvků lze taky snáze upravovat jejich vlastnosti (barva, velikost, reakce na vstupy uživatele). Při tomto způsobu vytváření GUI se generuje m-file obsahující funkční implementaci pro uživatelské rozhraní a soubor figure (soubor s příponou *.fig), který obsahuje dané GUI.

4.1 Tvorba GUI pro aplikaci

Při vytváření grafického uživatelského rozhraní pro aplikaci bylo použito obou způsobů tvorby GUI v MATLABu. Pomocí grafického editoru GUIDE se rozvrhly základní prvky a k ním se potom přidávala funkčnost během postupného vývoje aplikace. Přímého vytváření GUI se využilo tehdy, když bylo nutné reagovat na vstupy uživatele aplikace (například tvorba tabulek, dynamické vytvaření tlačítek, tvorba textových polí apod.).

Grafické uživatelské rozhraní se vytvářelo postupným vývojem, tedy na začátku byl jen jednoduchý nákres, jak by mělo výsledné GUI vypadat a dané prvky se přidávaly s rostoucí funkčností aplikace. Jelikož u této aplikace není grafické rozhraní nijak extrémně složité, tak se tento postup jevil jako velmi efektivní. U aplikací s rozsáhlejším GUI by bylo vhodnější si prvně vytvořit kompletně celý grafický návrh a k jednotlivým prvkům přidávat postupně funkce.

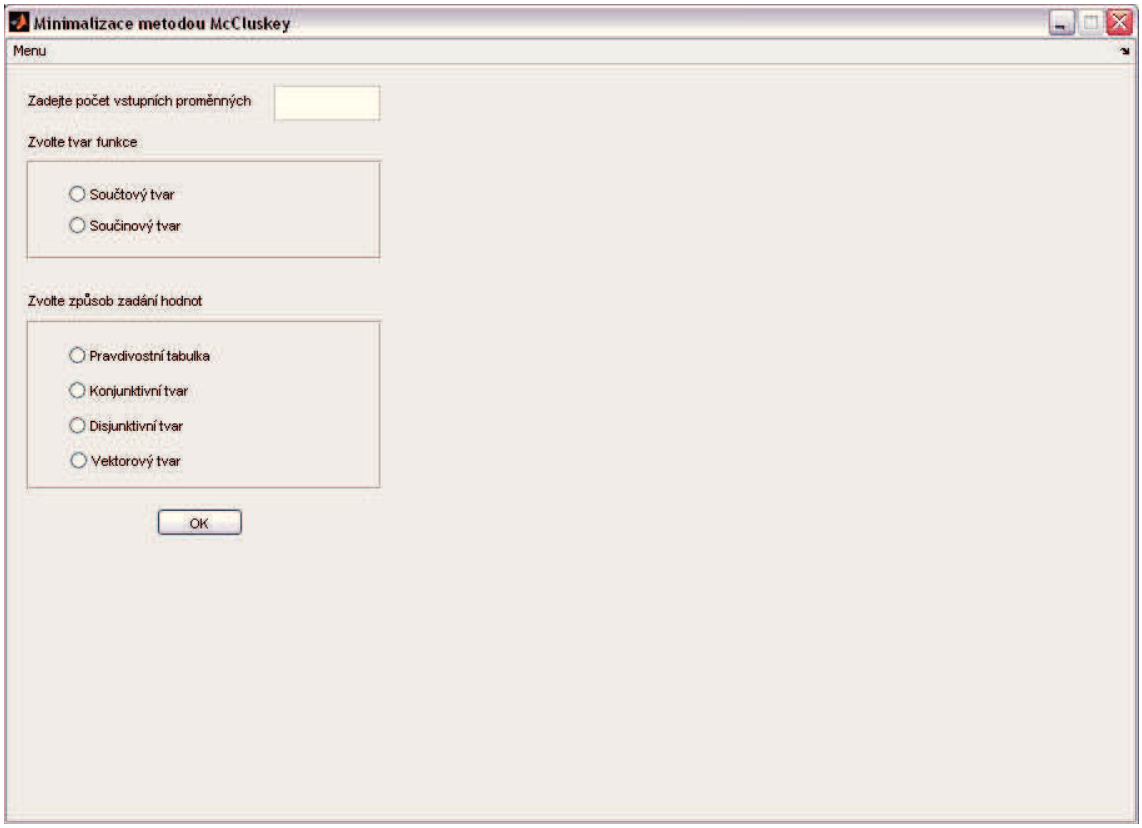
Celkovým cílem tvorby GUI bylo vytvoření uživetelsky přivětivého, přehledného a intuitivího grafického rozhraní, kde uživatel nebude tápat, jak s danou aplikaci pracovat.

4.2 Popis základního GUI a funkčnosti aplikace

Vstupní obrazovka aplikace se skládá ze tří základních částí - textového pole pro zadání počtu vstupních proměnných a dvou objektů pro zadání tvaru funkce a způsobu, jakým uživatel danou funkci zapíše. Pro tyto dva objekty bylo použito MATLAB objektu `uibuttongroup`, který má za úkol sdružovat `radiobuttony` do kontejneru, a tím umožnit pouze označení jednoho z `radiobuttonů` v daném `uibuttongroup`. Ukázka vstupní obrazovky je na obrázku 1.

Po zadání vstupních hodnot dojde k vykreslení obrazovky, která obsahuje dvě tabulky a slouží k minimalizaci zadané funkce. V levé tabulce má uživatel možnost označovat řádky tabulky za pomocí `checkboxů`, lze-li dva označené řádky spojit, tak se

hodnota po spojení řádku zapíše do pravé tabulky. Ve spodní části obrazovky se nachází tři tlačítka - tlačítko `Další`, které je neaktivní až do stavu, kdy dojde ke kompletnímu naplnění pravé tabulky a umožňuje pokračovat v další minimalizaci funkce, tlačítko `Nápověda`, které zobrazí uživateli, které dva řádky tabulky může spojit a tlačítko `Doplnit` sloužící k uplnému doplnění pravé tabulky.



Obrázek 1: Vstupní obrazovka aplikace.

4.2.1 Načtení vstupních dat

Načtení vstupních dat mají na starost funkce `edit1_Callback`, `uipanel4_SelectionChangeFcn` a `uipanel3_SelectionChangeFcn`. Funkce `edit1_Callback` získává hodnotu počtu vstupních proměnných funkce zadanou uživatelem z textového pole, kterou uloží do globální proměnné `pocet_sloupcu`, funkce `uipanel3_SelectionChangeFcn` určuje uživatelem označený tvar logické funkce a definuje globální proměnnou `tvar`, funkce `uipanel4_SelectionChangeFcn` a jí vytvořená globální proměnná `zadani` se stará o způsob vytvoření vložení dat podle volby. Po kliknutí na tlačítko `OK` na vstupní obrazovce aplikace, dojde k ověření zadaných vstupů. Pokud uživatel zapomene vyplnit počet vstupních hodnot nebo označit tvar či způsob zadání funkce, objeví se chybové hlášení. Je-li vše v pořádku vykreslí se objekty podle uživatelem zvoleného

způsobu zádání funkce (textový panel pro zadaní v disjunktivním, konjunktivním a vektorovém způsobu nebo pravdivostní tabulka pro zadnání hodnot pravdivostní tabulkou). U zadávání hodnot pomocí pravdivostní tabulky uživatel má na výběr ze tří hodnot (logická 0, logická I nebo X) u každého řádku. Poté po kliknutí na tlačítko `Další` dochází k uprávám zadaných hodnot.

4.2.2 Úpravy zadaných hodnot

Pokud se hodnoty zadají v disjuntivním, konjuktivním nebo vektorovém tvaru, tak se nejprve volá funkce `uprava_text_vstupu`, ve které se provede kontrola vstupního textového řetězce.

```
m = regexp(str, ' (((((\ d+\,)*+(\d+){1})?)+(\(((((\d+\,)*+(\d+){1}){1})\)){0,1}){1}', 'match');
chyba=0;
spatna=0;
if isempty(m)
    if isempty(str)
        messagebox('Nebyly zadány žádné vstupní hodnoty.');
    else
        messagebox('Chybně zadaný vstup.');
    end
    chyba=1; %tlacitko dalsi nevykona akci
else
```

Výpis 1: Část funkce uprava_text_vstupu pro úpravu hodnot zadaných v konjuntivním nebo disjunktivním tvaru

Vstupní textový řetězec se nejprve kontroluje pomocí regulárního výrazu. Pokud přes tuto kontrolu neprojde, vyvolá se okno s chybovou hláškou - pokud ve vstupu není nic zadáno, text chyby zní `'Nebyly zadány žádné vstupní hodnoty.'` jinak se vypíše `'Chybně zadaný vstup.'`.

```
if ~isempty(s{1})
    if zadani==2
        for i=1:length(s{1})
            data{1}{str2double(s{1}(1,i))+1,pocet_sloupcu+2}='0';
        end
    else
        for i=1:length(s{1})
            data{1}{str2double(s{1}(1,i))+1,pocet_sloupcu+2}='1';
        end
    end
end

if ~isempty(s{2})
    for j=1:length(s{2})
        data{1}{str2double(s{2}(1,j))+1,pocet_sloupcu+2}='X';
    end
end
```

Výpis 2: Ukázka plnění tabulky zadání ve funkci uprava_text_vstupu pro úpravu hodnot zadaných v konjuntivním nebo disjunktivním tvaru

Pokud podmínky zadaných hodnot jsou splněny, dojde k naplnění tabulky daty. Data na naplnění se vybírají ze dvou polí - podle toho zda byla funkce zadána úplně nebo neúplně. Cyklus prochází jednotlivé prvky polí a zapisuje je do struktury `data`, což je pole datových matic jednotlivých tabulek kroků minimalizace.

```
m = regexp(str,'\<[0-1X]+\>','match');
if isempty(m)
    if isempty(str)
        messagebox('Nebyly zadány žádné vstupní hodnoty.');
    else
        messagebox('Chybně zadaný vstup.');
    end
    chyba=1; %tlacitko dalsi nevykona akci
else
    if length(m{1})==zadana_hodnota
        for i=1:zadana_hodnota
            data{1}{i,pocet_sloupcu+2}=m{1}(i);
        end
    else
        if length(m{1})>zadana_hodnota
            messagebox('Zadáno více hodnot, než je povoleno.');
        else
            messagebox('Zadáno málo hodnot.');
        end
        chyba=1; %tlacitko dalsi nevykona akci
    end
end
```

Výpis 3: Ukázka kontroly a plnění tabulky zadání ve funkci uprava_text_vstupu pro úpravu hodnot zadaných ve vektorovém tvaru

Ošetření vstupu při zadání funkce ve vektorovém tvaru se kontroluje pomocí regulárního výrazu. Není-li vstup korektně zadán, je chyba v zadání hodnot zobrazena příslušnou chybovou hláškou uživateli. Při správném zadání dojde k naplnění struktury `data`.

4.2.3 Zpracování dat ve funkcích

Po zpracovaní zadaných dat se nejdříve musí upravit zadaná funkce. To má na starost m-file `prvni_uprava.m`, která se zavolá při prvním stisku tlačítka `Další`.

```
[~, pocet_log] = ismember(vstupni_cell(:,length(vstupni_cell (1,:) )),log_hodnota);
[~, pocet_X] = ismember(vstupni_cell(:,length(vstupni_cell (1,:) )),'X');
tabulka_pokryti=cell(1,2+sum(pocet_log)+sum(pocet_X));
tabulka_pokryti (1,1)={'F=I,X'};
tabulka_pokryti (1,length(tabulka_pokryti (1,:) ))={'Hodnoty'};
tabulka_pokryti (1,length(tabulka_pokryti (1,:) )+1)={int2str(sum(pocet_X))};
```

Výpis 4: Část funkce z prvni_uprava.m vytvářející tabulku pokrytí

Nejprve se ve funkci `prvni_uprava.m` vytvoří tabulka pokrytí, která bude důležitá při dalších krocích minimalizace. Spočítá se počet logických hodnot v zadané funkci,

případně (je-li funkce zadaná neúplně) počet hodnot X a vytvoří se základ tabulky pokrytí (vytvoří se datová struktura typu `cell` o vypočtené velikosti). Dále se do této struktury přidá sloupec `'Hodnoty'`, do kterého se během dalších kroků minimalizace budou zapisovat hodnoty jednotlivých implikantů a sloupec s celkovým počtem funkcí s hodnotou X.

```
pocet_X=0;
for i=1:length(vstupni_cell(:,1)),
    if strcmp(vstupni_cell(i-cislo_radku_sm,length(vstupni_cell(1,:))),log_hodnota_negace)==1
        vstupni_cell(i-cislo_radku_sm,:)=[];
        cislo_radku_sm=cislo_radku_sm+1;
    else
        if strcmp(vstupni_cell(i-cislo_radku_sm,length(vstupni_cell(1,:))),log_hodnota)==1
            tabulka_pokryti(1,1+i-cislo_radku_sm-pocet_X)=cellstr(vstupni_cell(i-cislo_radku_sm,1));
        end
        if strcmp(vstupni_cell(i-cislo_radku_sm,length(vstupni_cell(1,:))),'X')==1
            pocet_X=pocet_X+1;
            tabulka_pokryti(1,1+sum(pocet_log)+pocet_X)=cellstr(vstupni_cell(i-cislo_radku_sm,1));
        end
        [~, index] = ismember(vstupni_cell(i-cislo_radku_sm,2:length(vstupni_cell(1,:))-1),log_hodnota);
        vstupni_cell(i-cislo_radku_sm,length(vstupni_cell(1,:)))=cellstr(int2str(sum(index)));
    end
end

mCellArray=sortrows(vstupni_cell,length(vstupni_cell(1,:)));
[xa,xb,xc]=kontrola_tab(mCellArray,tabulka_pokryti,log_hodnota_neg);
```

Výpis 5: Část funkce z prvni_uprava.m upravující zadanou funkci

V cyklu se prochází jednotlivé řádky tabulky zadání funkce a odstraňují se ty vektory, které nabývají negované hodnoty k logické hodnotě tvaru zadání funkce. Zbylé názvy vektorů se zapíší do tabulky pokrytí (do první části tabulky pokrytí ty vektory, nabývající ve funkci logické hodnoty a do druhé části vektory s neurčitou logickou hodnotou. Poté se do posledního sloupce upravené tabulky zadání přidá součet logických hodnot u jednotlivých vektorů. Nakonec se tabulka setřídí podle počtu logických hodnot a zavolá se funkce v m-file `kontrola_tab.m`, která kontorluje, zda-li se v tabulce nachází implikanty.

Funkce `kontrola_tab` má tři vstupní parametry - tabulku hodnot, tabulku pokrytí a logickou hodnotu negace dané funkce. Nejprve se prochází jednotlivé řádky tabulky hodnot a v cyklu se zjišťuje, zda řádky splňují podmínku, aby rozdíl počtu logických hodnot byl roven 1. Pokud je tato podmínka splněna, tak dojde ke kontrole, zda se dané vektory liší pouze v jedné hodnotě. Pokud je i tato podmínka splněna, smažou se dané řádky z datové struktury `tab`, ve které jsou uloženy názvy jednotlivých řádků (první sloupec tabulky hodnot). Tento cyklus je zobrazen ve výpisu 6.

Zbydou-li v proměnné `tab` nějaké hodnoty, víme, že nedošlo u těchto hodnot ke splnění podmínek a jedná se tedy o implikanty. Toto obstarává řádek s funkcí `emptyCells = cellfun(@isempty,tab)`, která testuje v proměnné `tab` nějaká prázdná místa, pokud ano, ve výsledném vektoru `emptyCells` se zapíše hodnota 0, jinak se

zapíše hodnota 1. Pozice, na kterých se nachází hodnota 1 se uloží do proměnné `in2`, která se použije při zápisu implikantů do tabulky hodnot a pokrytí.

```
for i=1:length(c1(:,1))-1,
    vector_i=c1(i,1:length(c1(1,:))-1);
    for j=i+1:length(c1(:,1)),
        vector_l=vector_i;
        vector_j=c1(j,1:length(c1(1,:))-1);
        vector_l(1)={'0'};
        vector_j(1)={'0'};
        if ( str2double(c1(j,length(c1(1,:))))- str2double(c1(i,length(c1(1,:)))))==1
            if (sum(cellfun(@(vector_l,vector_j) vector_l~=vector_j, vector_l, vector_j)))==1
                [rn,cn]=find(strcmp(tab,c1{i,1}));
                tab(rn,cn)={[]};
                [rn2,cn2]=find(strcmp(tab,c1{j,1}));
                tab(rn2,cn2)={[]};
        end
     end
    end
end
```

Výpis 6: Ukázka části funkce kontrola_tab

Při procházení vektoru `in2` se k zápisu implikantu se využívá proměnná `impl`, jejíž hodnota je nastavena od začátku na 64. Pokud se má vytvořit implikant a zapsat do tabulek hodnot a pokrytí, tak se využije této proměnné a převede se pomocí příkazu `native2unicode` na textový znak, kterým se daný implikant označí.

```
vect=c1(in2(i),2:length(c1(1,:))-2);
if (strcmp(vect(e),log_hodnota))==1
    hodnota=strcat(hodnota,'*','x', int2str (length(vect)-e));
end
if (strcmp(vect(e),log_hodnota_negace))==1
    hodnota=strcat(hodnota,'*','-x', int2str (length(vect)-e));
end
.
.
.
c2(length(c2(:,1)),length(c2(1,:))-1)={strcat('(',hodnota,')')};
```

Výpis 7: Ukázka částí funkce kontrola_tab - zápis hodnot implikantů

Ve výpisu 7 je ukázana část kódu, kterým se vytváří a zapisují hodnoty implikantů do sloupce `Hodnoty` tabulky pokrytí pro funkci zadanou v součtovém tvaru. Ve vektoru, který je implikantem se hledají místa ve kterých nabývá buď logické hodnoty 1 nebo 0 a podle toho se vytváří výsledná hodnota implikantu.

Do tabulky pokrytí je poté nutné zapsat, které vektory daný implikant pokrývá a proto se rozdělí název vektoru pomocí příkazu `strread(str,'%u','delimiter', ', ')`. Tím nám vznikne pole `cisla`, ve kterém jsou uloženy názvy jednotlivých vektorů a podle nich se poté do tabulky pokrytí zapíše symbol X na místo, kde se název prvku pole shoduje s názvem sloupce - určí se, které vektory daný implikant pokrývá. Ukázka kódu ve výpisu 8.

```
cisla = strread(char(tab(in2(i),1)),'%u','delimiter',' ','␣');
for o=1:length(cisla),
   for p=2:length(c2(1,:))-2,
      if strcmp(int2str(cisla(o)),c2(1,p))==1
         c2(length(c2(:,1)),p)={'X'};
      end
   end
end
```

Výpis 8: Ukázka funkce kontrola_tab - zápis znaků X do tabulky pokrytí

Funkce `kontrola_tab` nakonec vrací tři proměnné a to - tabulku hodnot (nedoplněnou případně doplněnou o implikanty), tabulku pokrytí a hodnotu -1 (pokud nejsou všechny řádky tabulky implikanty) nebo 1 (pokud naopak jsou všechny řádky tabulky implikanty, a tím pádem se nevykonávájí další kroky minimalizace logické funkce.)

4.2.4 Označení řádků tabulky uživatelem

Uživatel má možnost označovat řádky tabulky, které chce spojit. O to se starají tři funkce: `Oznaceni_Callback`, `zapis_radek` a m-file `porovnaniradku.m`.

```
radek=e.Indices(1);
zapis_radek(radek);
```

Výpis 9: Ukázka funkce Oznaceni_Callback

Tato funkce se stará o získání čísla označeného řádku tabulky, který je poté vstupním parametrem funkce `zapis_radek`.

```
if pocet_clicku==0
   rad_1=c_radku;
  pocet_clicku=pocet_clicku+1;
else
    if rad_1==c_radku
    else
       h = findobj('Tag','druha_tab');
       xx=get(h,'Data');
       if ~isempty(xx)
          vstupni_cell3=xx;
       else
          vstupni_cell3={};
       end
       [xa,xb]=porovnaniradku(data{counter},rad_1-1,c_radku-1,log_hodnota_negace,0,
            vstupni_cell3)
```

Výpis 10: Ukázka části funkce zapis_radek

Číslo řádku, které vstoupí do funkce `zapis_radek` se uloží do proměnné `rad_1`. Po označení dalšího řádku se porovná, zda se liší od proměnné `rad_1`. Pokud ano, tak se nejprve do proměnné `xx` načtou data z pomocné tabulky a zavolá se funkce `porovnaniradku`, která navrací dvě proměnné.

```
if (strcmp(xa{1},'-2'))
   messagebox('Oznacene radky nesplnuji podminku, ze lze spojit pouze radky, ktere se lisi
       pouze v jedné logické hodnotě');
elseif (strcmp(xa{1},'-1'))
   messagebox('Oznacene radky nesplnuji podminku, ze lze spojit pouze radky, v nichz je rozdil
       poctu logickych hodnot roven 1');
elseif (strcmp(xa{1},'-3'))
else
   xx=[xx;xa];
   set(h,'Data',xx);

   if (strcmp(xb,'1'))==1
      h = findobj('Tag','dalsi');
      set(h,'Enable','on');
      k= findobj('Tag','doplnit');
      set(k,'Enable','off');
      j = findobj('Tag','napoveda');
      set(j,'Enable','off');
   end
end

h1 = findobj('Tag','tabulka');
dat_tab{1,1}=get(h1,'Data');
if (length(dat_tab{1}{1,length(dat_tab)})==1)
   dat_tab{1}(:,length(dat_tab{1}(1,:)))={false};
   set(h1,'Data',dat_tab{1});
else
   dat_tab{1}{1,length(dat_tab)}(:,length(dat_tab{1}{1,length(dat_tab)}(1,:)))={false};
   set(h1,'Data',dat_tab{1}{1,length(dat_tab)});
end
```

Výpis 11: Ukázka části funkce zapis_radek návratové hodnoty z funkce `porovnaniradku`

Ve výpisu 11 je ukázán kód z funkce `zapis_radek`, který se stará o zpracovnání návratových hodnot z m-filu `porovnaniradku`. Je-li návratová hodnota -1 nebo -2, uživatel označil řádky, které nesplňují podmínku spojení, a proto se vypíše příslušná chybová zpráva. Při návratové hodnotě -3, uživatel označil řádky, které už byly jednou spojeny, a proto se neprovede žádná akce. V jiném případě došlo ke spojení a výsledný vektor se přidá do druhé tabulky. Nastane-li situace, že se jedná o poslední přidaný řádek do druhé tabulky (v tabulce se nachází všechny možné spojené řádky) odblokuje se tlačítko `Další`. Nakonec dojde k odznačení checkboxů, sloužících k označovaní vektorů v tabulce.

Funkce `porovnaniradku` má celkem šest vstupních parametrů - tabulku dat, čísla označených řádků, logickou hodnotu negace funkce, podmínku porovnání a hodnoty dat z pomocné tabulky. Podmínka porovnaní může nabývat tří hodnot $(0,1,2)$. Hodnoty 0 nabývá, pokud se uživatel označuje checkboxy u tabulky. Kód vychází z toho, že se musí tak jako ve výpisu 6 porovnat vektory, zda splňují podmínky spojení. Splňují-li tyto podmínky, hodnota kde se liší se nahradí znakem "-" a spočítá se počet logických hodnot ve vektoru a výsledný vektor se navrátí zpět. Ještě před vrácením se ověří, zda se

nejedná o poslední možné spojení vektorů. Ze vstupní proměnné, která reprezentuje tabulku dat se vytvoří tabulka, reprezentující všechny možnosti spojení vektorů. Poté se porovná, zda-li jsou si rovny - pokud ano, druhá návratová hodnota funkce bude 1, jinak se navrací -1. Ukázka algoritmu je zobrazena ve výpisu 12.

```
for i=1:length(vstupni_cell(:,1))-1,
    vector_i=vstupni_cell(i,1:length(vstupni_cell(1,:))-1);
    for j=i+1:length(vstupni_cell(:,1)),
        vector_l=vector_i;
        vector_j=vstupni_cell(j,1:length(vstupni_cell(1,:))-1);
        vector_l(1)={'0'};
        vector_j(1)={'0'};
        if(str2double(vstupni_cell(j,length(vstupni_cell(1,:))))-str2double(vstupni_cell(i,length(
            vstupni_cell(1,:)))))==1
            if (sum(cellfun(@(vector_l,vector_j) vector_l~=vector_j,vector_l,vector_j)))==1
                vector_l(find(cellfun(@(vector_l,vector_j) vector_l~=vector_j,vector_l,vector_j),1)) =
                    {'-'};
                vector_l(1)={strcat(vstupni_cell{i,1},',',vstupni_cell{j,1})};
                [~,index] = ismember(vector_l,log_hodnota);
                vector_l(length(vector_l)+1)={int2str((sum(index)))};
                a=[a;vector_l];
            end
        end
    end
end

for i=1:length(a(:,1))-1,
    pocet_sm_rad=0;
    for j=i+1:length(a(:,1)),
        if strcmp(a(i,length(a(1,:))),a(j-pocet_sm_rad,length(a(1,:))))==1
            if strcmp(a(i,2:length(a(1,:))),a(j-pocet_sm_rad,2:length(a(1,:))))==1
                a(j-pocet_sm_rad,:) = [];
                pocet_sm_rad=pocet_sm_rad+1;
            end
        end
    end
end

if (vstupni_podminka==0)
    if (length(vstupni_cell_1(:,1))~=length(a(:,1)))
        vysl(1)={'-1'}; % neodblokuje tlačítko -> není kompletně doplněna tab
    else
        vysl(1)={'1'};  % odblokuje tlačítko -> kompletně doplněna tab
    end
```

Výpis 12: Ukázka funkce `porovnaniradku.m`

Zbylé dvě hodnoty podmínky porovnání se pooužívají při akcích na tlačítka `Nápověda` a `Doplnit`. Pokud uživatel klikne na tlačítko `Doplnit`, tak se zavolá funkce `doplnit_Callback`, která vyvolá funkci `porovnaniradku`, přičemž podmínkou porovnání je v tomto případě hodnota 2. Funkce potom navrací tabulku všech možností spojení (viz. výpis 12) a jelikož je doplněním tabulky splněn krok minimalizace, tak znovu dojde k odblokování tlačítka `Další`.

Při kliknutí na tlačítko `Nápověda` funkce `napoveda_Callback` se využívá znovu `porovnaniradku` tentokrát je podmínkou porovnaní hodnota 1. Ve funkci `porovnaniradku` se porovnájí hodnoty z druhé tabulky s tabulkou všech možností spojení. Pokud se narazí na vektor, který se tam nachází, tak se vektor z tabulky všech možností spojení smaže. Ze zbylých vektorů se potom vybere ten na prvním místě a je doporučen uživateli ke spojení. Ukázka kódu pro podmínku porovnaní 1 nebo 2 ve výpisu 13.

```
elseif (vstupni_podminka==1)
    pocet_sm_rad=0;
    for i=1:length(a(:,1))
        vect1=a(i-pocet_sm_rad,2:length(a(1,:)));
        for j=1:length(vstupni_cell_1 (:,1) )
            vect2=vstupni_cell_1 ( j ,2:length(vstupni_cell_1 (1,:) ) );
            if strcmp(vect1,vect2)==1
                    a(i-pocet_sm_rad,:)=[];
                    pocet_sm_rad=pocet_sm_rad+1;
                    break;
            end
        end
    end
    if (~isempty(a))
        nap=strfind(a(1,1), ' , ' );
        pozice_carky=round((length(nap{1}))/2);
        l = nap{1}(pozice_carky);
        cis1=a{1,1}(1:l-1);
        cis2=a{1,1}(l+1:length(a{1,1}));
        vector1(1)={cis1};
        vector1(2)={cis2};
        vysl={'-1'};
    else
        vector1={};
        vysl={'1'};
    end
elseif (vstupni_podminka==2)
    vector1=a;
    vysl(1)={'1'};
end
```

Výpis 13: Ukázka části funkce `porovnaniradku` pro podmínku porovnání 1 a 2

Po kompletním označení všech možností uživatelem a tedy naplnění druhé tabulky dojde k odblokování tlačítka `Další`. Při kliknutí na tlačítko `Další` dojde k zavolání funkce `setridenitabulky`, která má za úkol setřídit data v tabulce, protože uživatel může označovat řádky v náhodném pořadí a zjistit, zda se v druhé tabulce nenachází nějaký implikant. Kvůli tomu se po setřídění volá funkce `kontrola_tab`, tak jako u funkce `prvni_uprava`. Ukázka funkce je ve výpisu 14.

```
function [xa,xb,xc] = setridenitabulky ( vstupni_cell2 ,tab_pok,log_hodnota_neg)
mCellArray=sortrows(vstupni_cell2,length(vstupni_cell2 (1,:) ) );
[xa,xb,xc]=kontrola_tab(mCellArray,tab_pok,log_hodnota_neg);
end
```

Výpis 14: Funkce setridenitabulky.m

4.2.5 Vytvoření minimalizované funkce

Po provedení všech kroků minimalizace se musí vytvořit výsledek minimalizované funkce z tabulky pokrytí. K tomu složí m-file `vysledek_tab.m`, který má dva vstupní paramatery - tabulku implikantů a logickou hodnotu negace funkce.

```
delka_tab=length(tab1(:,1));
delka_tab2=length(tab1(1,:));
for i=2:delka_tab2-2-str2double(tab1{1,delka_tab2});
    vect1=tab1(:,i);
    f=@(str)isequal(str, 'X');
    cislo = sum(cellfun(f,vect1));
    tab1(delka_tab+1,i)={cislo};
end
for i=2:delka_tab
    vect1=tab1(i,2:delka_tab2-2-str2double(tab1{1,delka_tab2}));
    vect2=tab1(i,2:delka_tab2-2);
    f=@(str)isequal(str, 'X');
    cislo = sum(cellfun(f,vect1));
    cislo2=sum(cellfun(f,vect2));
    tab1(i,delka_tab2)={cislo};
    tab1(i,delka_tab2+1)={cislo2};
end
```

Výpis 15: Ukázka části funkce vysledek_tab - spočítání jednotlivých symbolů X v řádcích a sloupcích

Ve výpisu 15 je kód, který počítá počet symbolů X v jednotlivých sloupcích a řádcích tabulky pokrytí. V prvním cyklu `for` se spočítá počet symbolů X v jednotlivých slupcích, ve druhém cyklu pak v řádcích. V řádcích se nejprve počítá součet symbolů X v první části tabulky a potom pro celý řádek. Hodnota počtu první části je vhodná při minimalizaci abychom určili, který z implikantů pokrývá nejvíce vektorů. Součet celého řádku se využije, pokud se budeme rozhodovat mezi více implikanty, kteří pokrývají stejné vektory. Poté se volí ten implikant, který pokrývá v celé tabulce více vektorů.

```
vect3=tab1(delka_tab+1,1:length(tab1(1,:))-3-str2double(tab1{1,length(tab1(1,:))-1}));
vys=cell(1,1);
while(sum(cellfun(@(str)isequal(str, 1),vect3))>0)
    o~= find(cellfun(@(str)isequal(str, 1),vect3),1);
    f=@(str)isequal(str, 'X');
    c_r = find(cellfun(f,tab1(:,o)),'X');
    vys{1,1}(length(vys{1,1})+1)=tab1(c_r,1);
    d=find(cellfun(f,tab1(c_r,1:length(tab1(1,:))-3-str2double(tab1{1,length(tab1(1,:))-1}))),'X');
    sm=0;
    for i=1:length(d)
        tab1(:,d(i)-sm)=[];
        sm=sm+1;
    end
    vect3=tab1(delka_tab+1,1:length(tab1(1,:))-3-str2double(tab1{1,length(tab1(1,:))-1}));
end
```

Výpis 16: Ukázka části funkce vysledek_tab - určení nevyhnutelných implikantů

V této části kódu se nejprve ve vstupní tabulce pokrytí hledá sloupec, kde je celkový součet symbolů X roven 1 (tzv. nevyhnutelný implikant). Do proměnné `vect3` se uloží poslední řádek tabulky pokrytí, který po předchozí úpravě obsahuje počet znaků X v jednotlivých sloupcích tabulky. V cyklu `while` se v případě, že v tabulce pokrytí se nachází nějaký nevyhnutelný implikant, zapiší dané implikanty do proměnné `vys`. Nejprve se zjistí, ve kterém sloupci se nachází pouze jeden znak X, poté řádek, na kterém se nachází daný znak nachází a zapíše se název implikantu do proměnné `vys`. Nakonec se z tabulky pokrytí odstaní veškeré sloupce, které pokrývá daný implikant. Cyklus `while` se ukončí až se v tabulce pokrytí nevyskytuje žádný nevyhnutelný implikant.

```
MyCell=cell2mat(tab1(2:length(tab1(:,1)),length(tab1(1,:))-1));
[bn,~]=find(MyCell==max(MyCell));
if (length(bn)==1)
    vys{1,1}(length(vys{1,1})+1)=tab1(bn+1,1);
    d=find(cellfun(f,tab1(bn+1,1:length(tab1(1,:))-3-str2double(tab1{1,length(tab1(1,:))-1})),'X'
        )
    sm=0;
    for i=1:length(d)
        tab1(:,d(i)-sm)=[];
        sm=sm+1;
    end
else
    MyCell2=cell2mat(tab1(2:length(tab1(:,1)),length(tab1(1,:))));
    for k=1:length(MyCell2)
        if k~=bn
            MyCell2(k)=0;
        end
    end
    [bn2,~]=find(MyCell2,max(MyCell2)); %radky, kde se nachazi maxima
    sum1=0;
    sum2=0;
    for a=1:length(bn2)
        sum1=sum1+length(bn2)-a;
        vect_pr=tab1(bn2(a)+1,2:length(tab1(1,:))-3-str2double(tab1{1,length(tab1(1,:))-1}));
        for b=(a+1):length(bn2)
            vect_pr2=tab1(bn2(b)+1,2:length(tab1(1,:))-3-str2double(tab1{1,length(tab1(1,:))-1}));
            if isequal(vect_pr,vect_pr2)
                sum2=sum2+1;
            end
        end
    end
    if (sum1~=sum2)
        bn2=bn2(1);
    end
```

Výpis 17: Ukázka části funkce vysledek_tab - určení výsledných implikantů

Ve výpisu 17 dochází k výběru dalších implikantů pro výslednou minimalizovanou funkci. Tentokrát se volí implikanty, které pokrývají nejvíce vektorů. Vybere se ten řádek, obsahující nejvíce symbolů X v první části tabulky. Mohou nastat dvě situace - maximální hodnoty nabývá jen jeden řádek nebo maximalní hodnoty nabývá řádků více. V prvním případě se implikant zapíše k výsledku a vektory, které pokrývá se z tabulky pokrytí

odstraní. V druhém případě je pak nutné rozhodnout, který z implikantů pokrývá více vektorů v celé tabulce pokrytí. Proto je nutné k porovnání použít poslední sloupec tabulky pokrytí, který v sobě uchovává celkový počet symbolů X v daném řádku. Ale i zde může nastat situace, že dané řádky pokrývají stejný počet vektorů. Musíme tedy porovnat, zda-li implikanty nepokrývají stejné zbylé vektory (bylo by více výsledků). K tomu slouží cyklus `for`, který porovnává vektory mezi sebou. Pokud shodné nejsou vybere se z těch vektorů ten na prvním místě. Jsou-li všechny shodné, vytvoří se více výsledků s řešením.

```
sm=0;
vys1=vys{1,1};
for i=1:length(bn2)
    if (length(vys1)>1)
        vys{1,i}=vys{1,1};
        vys{1,i}(length(vys1)+1)=tab1(bn2(i)+1,1);
    else
        vys{1,i}(length(vys1)+1)=tab1(bn2(i)+1,1);
    end
    f=@(str)isequal(str, 'X');
    d=find(cellfun(f,tab1(bn2(i)+1,1:length(tab1(1,:))-3-str2double(tab1{1,length(tab1(1,:))-1}))
        ,'X');
    if (length(tab1(2,:))-4-str2double(tab1{1,length(tab1(1,:))-1}))>0
        tab1(:,d-sm)=[];
        sm=sm+1;
    end
end
```

Výpis 18: Ukázka části funkce vysledek_tab - přidaní implikantů do výsledku

Cyklus `for` ve výpisu 18 prochází implikanty s maximy hodnot v posledním sloupci tabulky pokrytí a přidává je k výsledku. Nakonec zase dochází ke smazaní vektorů z tabulky pokrytí, které daný implikant pokrývá.

```
for u=1:length(vys)
    vys_txt=strcat('F',int2str(u),'␣=/');
    vys_txt2='';
    for v=1:length(vys{u})
        f1=char(vys{u}(1,v));
        f=@(str)isequal(str, f1);
        cislo = find(cellfun(f,tab1(:,1)),f1);
        if (v==length(vys{u}))
            vys_txt=strcat(vys_txt,vys{u}(1,v),'␣=␣');
            vys_txt2=strcat(vys_txt2,tab1(cislo,length(tab1(1,:))-2));
        else
            vys_txt=strcat('/',vys_txt,vys{u}(1,v),znamenko,'/');
            vys_txt2=strcat('/',vys_txt2,tab1(cislo,length(tab1(1,:))-2),znamenko,'/');
        end
    end
vys(1,u)=strrep(strcat(vys_txt,vys_txt2),'/','␣');
end
```

Výpis 19: Ukázka části funkce vysledek_tab - zápis výsledku

Zápis výsledku probíhá jednoduše, prochází se hodnoty v proměnné `vys`, kde jsou uloženy názvy jednotlivých implikantů minimalizované fuknce. Podle jejich názvu se naleznou a zapíší hodnoty jim náležící z tabulky pokrytí. Během tohoto tvoření textových řetězců s výslednou minimalizovanou funkcí, se přidává znak "/" k jednotlivým řetězcům a to pouze z estetického hlediska, jelikož při funkci spojovaní řetězců `strcat` MATLAB ořezává mezery na začátku a koncích řetězce. Po vytvoření výsledného řetězce je potom znak "/" nahrazen mezerou a řetězec se stává pro uživatele lépe čitelným.

4.2.6 Zobrazení výsledné minimalizované funkce

Výsledek minimalizace se zobrazuje ve víceřádkovém edit-boxu. Problém u Matlabu je, že v normálním víceřádkovém edit-boxu se nezobrazují některé speciální znaky, proto je nutné v tomto případě přistupovat ke GUI prvku na nižší úrovni pomocí externí funkce `findjobj.m` [4]. Použití této funkce nám dovolí využívat vlastností Java grafických objektů, tedy i podpory HTML znaků, které jsou součástí textového řetězce s obsahem minimalizované funkce.

```
hEditbox = uicontrol('style','edit','max',10,'Parent',panel_tab,...
                'Position',[10  10 845 400],'Tag','hEditBox');
jScrollPane = findjobj(hEditbox);
jViewPort = jScrollPane.getViewport;
jEditbox = jViewPort.getComponent(0);
jEditbox.setEditable(false);
jEditbox.setContentType('text/html');
final ='';
for i=1:length(vysl_text)
    final =strcat(final,'<p>',strrep(vysl_text(1,i),'-','&not'),'</p>');
end
jEditbox.setText(final);
t=jEditbox;
```

Výpis 20: Zobrazení výsledku v GUI prvku Editbox

Nejprve se vytvoří víceřádkový edit-box s názvem `hEditbox`, jelikož byla použita funkce `findjobj`, tak na `hEditbox` lze aplikovat funkčnost Java objektů (například `setContentType('text/html')`, kterým se nastaví druh obsahu textového pole na podporu HTML znaků). Ale i Java nepodporuje všechny CSS deklarace. V tomto případě nedokáže interpretovat `text-decoration: overline`, tedy nadtržení textu (v tomto smyslu jako symbol negace) by se nezobrazilo. Aby se této chybě předešlo, bylo ve výsledku vrchní nadtržení nahrazeno symbolem negace ¬ příkazem `strrep(vysl_text-(1,i),'-',' &not ')`.

5 Export GUI pomocí MATLAB Java Builderu

Šíření aplikací napsaných v MATLABu je možné několika způsoby. Jedním z nich je přímé stáhnutí m-filů (aplikace) k uživateli, ale tento způsob vyžaduje, aby uživatel měl na počítači nainstalovaný buď MATLAB nebo MCR. Dalším způsobem je mít nahranou aplikaci na MATLAB Web Serveru. Uživatel poté zadává hodnoty do formulářů HTML stránek a CGI scripty komunikují s těmito daty a aplikací na MATLAB Web Serveru. Třetí možností je vytvoření webové komponenty, která obstarává spolupráci s napsanou aplikací. Nevýhodou posledních dvou způsobů je nemožnost využít MATLAB GUI [6].

Jelikož Matlab server, který by měl být na katedře telekomunikační techniky Vysoké školy Báňské - Technické univerzity Ostrava, využívá Java komponent, byl zvolen pro šíření aplikace způsob vytvoření webové komponenty (applet, servlet) v programovacím jazyce Java. Pro tento způsob vývoje webové aplikace je nutné mít nainstalovaný kromě MATLABu také MATLAB Compiler a MATLAB Builder JA.

5.1 Použité nástroje a komponenty

5.1.1 MATLAB Compiler

Tento nástroj slouží k vytváření komponent z MATLABu, které mohou být využity externími aplikacemi. Tímto způsobem se dá distribuovat MATLAB kód i pro aplikace využívající Microsoft Excel, .NET a Java komponenty. Pro každou z těchto aplikací je nutné použít jiný MATLAB Builder - pro Excel add-ins MATLAB Builder EX, pro .NET komponenty MATLAB Builder NE a pro Java komponenty MATLAB Builder JA [5].

5.1.2 MATLAB Builder JA

Hlavní funkcí tohoto Builderu je vytváření Java tříd z m-filů. Výsledné třídy mohou být využívany programátorem v Java aplikacích (jak klasických, tak webových). Postup pro práci s MATLAB Builder JA je popsán v části 5.2.1.

5.1.3 Applet

Appletem nazýváme Java aplikaci, která je umístěna na webové stránce, kde ji lze spouštět. Java třída appletu musí být potomkem třídy `java.applet.Applet`, v případě užití grafické knihovny Swing `javax.swing.JApplet`. Užitím appletu umožňujeme uživateli větší interaktivnost, jelikož pomocí jazyku appletu je možné do webové stránky umístit složitější grafické prvky nebo můžeme například přehrávat video či audio formáty, které nejsou ze základu podporovány prohlížečem. O spuštění appletu na webové stránce se stará JVM prohlížeče.

5.1.4 Servlet

Jedná se o Java třídu, která se nejčastěji využívá ke generování dynamických webových stránek. Servletem rozumíme Java třídu, která je potomkem třídy `javax.servlet.`-

`http.HttpServlet`. Samotný servlet je založen na principu požadavek-odpověď (request-response), kdy získává požadavky z HTML stránek (většinou z formulářů, appletů), poté vykoná kód (ať už sám nebo odešle daný požadavek dále - do databáze, případně do jiné aplikace) a vratí odpověď zpět uživateli.

5.1.5 MWArray

Tato abstraktní třída má na starost převod datových typů mezi Javou a MATLABem. Je součástí API z MATLAB Builder JA a k implementaci se používá balík `com.mathworks-.toolbox.javabuilder.MWArray`. MWArray obsahuje několik podtříd, které poskytují konstruktory a metody pro vytvoření hlavních MATLAB datových polí z Java objektů. Mezi tyto podtřídy patří:

- MWNumericArray,
- MWLogicalArray,
- MWCharArray,
- MWCellArray,
- MWStructArray [7].

Práce s abstarktní třídou MWArray je uvedena v části 5.2.3.

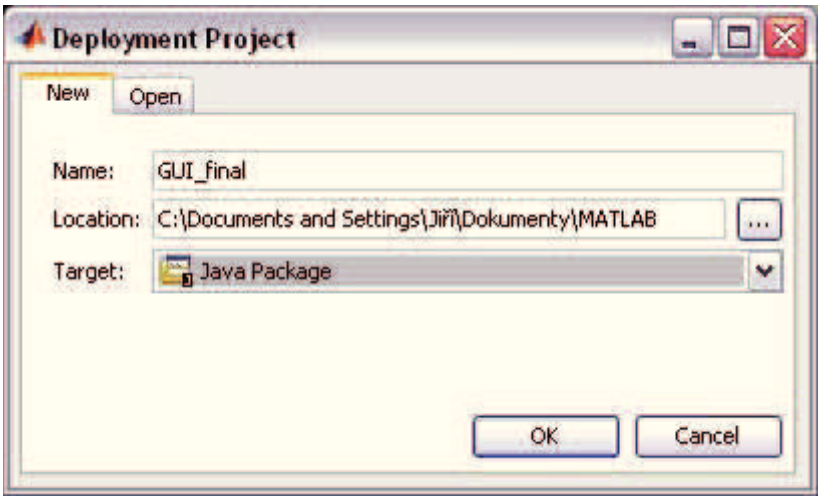
5.2 Ukázka tvorby aplikace pro Matlab server

Aby byl omožné využít aplikace na Matlab serveru je nejdříve nutné vytvořit Java třídy z m-filů, poté naprogramovat GUI appletu a zprovoznit komunikaci mezi appletem a servletem.

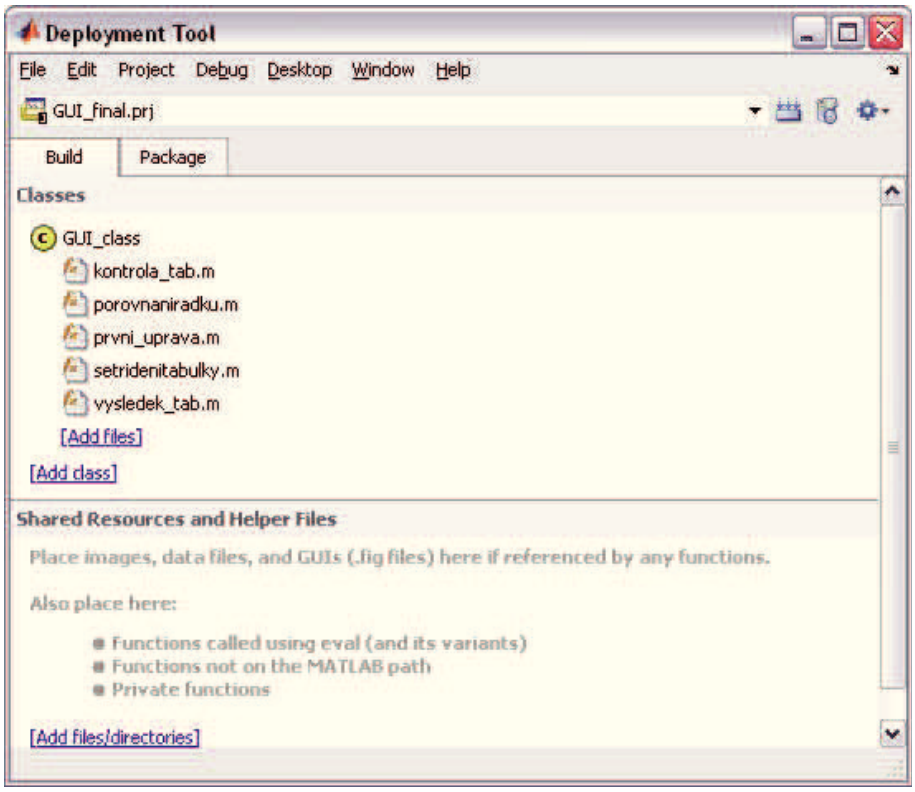
5.2.1 Vytvoření Java tříd

Pro vytvoření Java tříd z m-filů se využívá nástroj, který se spouští z MATLABu příkazem `deploytool`. Po spuštění se objeví vstupní obrazovka (viz. obrázek 2), ve které do pole `Name` zadáme název projektu (v našem případě `GUI_final`) a v poli `Target` vybereme řádek `Java Package` pro vytvoření Java tříd. Poté stačí kliknout na tlačítko `OK` a dojde k vytvoření projektu.

Pro přidání m-filů aplikace do projektu se musí vytvořit Java třída kliknutím na text `Add class`, kterou v našem případě nazveme `GUI_class`. Do této třídy se přidají jednotlivé m-fily kliknutím na text `Add files`. Po přidaní všech potřebných m-filů, stačí spustit samotný proces vytvoření buď kliknutím na tlačítko `Build`, které se nachází vedle pole pro název projektu nebo můžeme toto akci vyvolat přes menu `Project - Build`. Ukázka je vyobrazena na obrázku 3.



Obrázek 2: Úvodní obrazovka nástroje `deploytool`



Obrázek 3: Přidávání m-filů do Java třídy v nástroji `deploytool`

5.2.2 Komunikace mezi appletem a servletem

Jelikož applet nemůže použít napsané GUI z MATLABu, je nutné si ho naprogramovat v Jave. Vytvořilo se co nepodobnější GUI v rámci možností, takže vzhled i ovladaní aplikace je velmi podobný s MATLAB GUI a liší se jen v některých částech - například pro zobrazovaní tabulek jednotlivých kroků minimalizace bylo využito v appletu záložek (jsou standardně implementovány v knihovně Java Swing, použitého pro tvorbu GUI) místo tlačítek v MATLABu. Po funkční stránce je ale nejdůležitější samotná komunikace mezi appletem a servletem, proto nebude samotné GUI více popsáno.

U appletu je nejnutnější vytvoření připojení k servletu, odeslání dat na servlet a jejich získání zpět.

```
private URLConnection getServletConnection() throws MalformedURLException,IOException {
    URL urlServlet = new URL(getCodeBase(),"applet");
    URLConnection con = urlServlet.openConnection();
    con.setDoInput(true);
    con.setDoOutput(true);
    con.setUseCaches(false);
    con.setRequestProperty("Content-Type","application/x-java-serialized-object");
    return con;
}
```

Výpis 21: Metoda `getServletConnection()`

V metodě `getServletConnection()` se získá URL adresa na servlet, který podle zápisu URL `urlServlet = new URL(getCodeBase(),"applet")` je uložen na serveru ve složce `applet`. Poté se otevře připojení, které bude se servletem komunikovat a nastaví se mu vlastnosti (pomocí `con.setDoInput(true)` a `con.setDoOutput(true)` se stanoví, že applet může data odesílat i přijmat, `con.setUseCaches(false)` zamezí využívaní vyrovnávací paměti a kód `con.setRequestProperty("Content-Type","application/x-java-serialized-object")` určuje formát odesílaných dat jako serializovaný java objekt.

```
URLConnection con = getServletConnection();
OutputStream outstream = con.getOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(outstream);
Object[] data_send;
 ...
oos.writeObject(data_send);
oos.flush();
oos.close();

InputStream instr = con.getInputStream();
ObjectInputStream inputFromServlet = new ObjectInputStream(instr);
Object[] result = (Object[]) inputFromServlet.readObject();
inputFromServlet.close();
instr.close();
```

Výpis 22: Odeslání a příjem dat u appletu

Pro odesílaní dat, se po vytvoření připojení na servlet naplní datový proud typu `ObjectOutputStream` daty (jelikož se posílá více proměných, tak jsou uloženy do pole

typu `Object`) a odešle servletu. Po zpracovaní dat servletem, se data které vrací přijmají pomocí datového proudu `ObjectInputStream` a vytvoří se znovu pole typu `Object`, kde se data uloží a applet je poté dále zpracovává. Velmi obdobně funguje tento příjem a odeslání dat na straně servletu. Jen se musí nastavit formát vystupních dat (znovu na serializovaný objekt), které bude servlet navracet appletu. Ukázka je ve výpisu 23.

```
response.setContentType("application/x-java-serialized-object");
InputStream in = request.getInputStream();
ObjectInputStream inputFromApplet = new ObjectInputStream(in);
Object[] input = (Object[]) inputFromApplet.readObject();
...
Object[] out=new Object[3];
OutputStream outstr = response.getOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(outstr);
oos.writeObject(out);
oos.flush();
oos.close();
```

Výpis 23: Příjem dat servletem a odeslání výsledku appletu

5.2.3 Předávání dat mezi Javou a MATLABem

Aby bylo možné předávat datové typy Javy do MATLABu, je nutné jejich přetypovaní do specifického pole podtřídy MWArray.

```
Object [][] echo = (Object [][]) input[1];
int dl_pole = Integer.parseInt(input[3].toString());
MWCellArray cislo = new MWCellArray(echo.length, echo[0].length);
for (int i = 1; i <= echo.length; i++) {
    for (int j = 1; j <= dl_pole; j++) {
        cislo.set(new int[]{i, j}, echo[i-1][j-1].toString());
    }
}
GUI_class gui_class = new GUI_class();
Object[] navrat = gui_class.prvni_uprava(3, cislo, new MWNumericArray(tvar));
```

Výpis 24: Předavání datových typů mezi Javou a MATLABem

Ve výpisu 24 je ukázano, jak se přijatá data z appletu přetypovávájí a odesílají do MATLABu. Do dvojrozměrného pole `echo` se uloží data z appletu obsahující pravdivostní tabulku zadání funkce. Kvůli možnosti odeslání do MATLABu se nejprve nadeklaruje pole `cislo` podtřídy `MWCellArray`, která reprezentuje datovou strukturu `cell` v MATLABu a poté se pomocí cyklů `for` naplní. Abychom mohli přistupovat k m-filům, vytvoříme instanci třídy `GUI_class` a kódem `Object[] navrat=gui_class.prvni_uprava(3,cislo,new MWNumericArray(tvar))` se volá funkce `prvni_uprava`. Prvním parametrem volané funkce je vždy počet návratových hodnot (v tomto případě se jedná o tři), dalšími jsou klasicky vstupní parametry funkce (u m-filu `prvni_uprava` se jedná o pravdivostní tabulku zadání funkce, kterou reprezentuje proměnná `cislo` a logickou hodnotu negace funkce, která je uložena v proměnné `tvar`. I tuto hodnotu je nutné přetypovat na `MWNumericArray` pomocí příkazu `MWNumericArray(tvar)`).

```
MWCellArray x1 = (MWCellArray) navrat[0];
int [] diim = x1.getDimensions();
String [][] matice = new String[diim[0]][diim[1]];
for (int i = 1; i <= diim[0]; i++) {
    for (int j = 1; j <= diim[1]; j++) {
        matice[i-1][j-1] = x1.getCell(new int[]{i, j}).toString();
    }
}
```

Výpis 25: Převod navratových hodnot z MATLABu

Při získávání dat z MATLABu je nutné data převést do datových typů Javy. Ve výpisu 25 jsou v proměnné `navrat` uloženy navratové hodnoty MATLAB funkce, která navrací v tomto případě data v datové struktuře MATALBu typu `cell`. Z tohoto důvodu se první prvek pole `navrat` přetypuje kódem `MWCellArray x1 = (MWCellArray) navrat[0]` na datový typ `MWCellArray`. Do proměnné `diim` se metodou `x1.getDimensions()` uloží rozměry (počet řádku a počet sloupců) daného prvku pole, jejíž hodnoty se využijí při deklaraci dvojrozměrnného pole datového typu `String`, které se poté naplní pomocí dvou cyklů `for`, ze kterých se metodou `x1.getCell(new int[]i, j)` získavají data z návratové hodnoty typu `cell`. Pokud by byla hodnota typu `MWNumericArray`, tak se použije metoda `getDouble`.

5.2.4 Nasazení appletu a servletu na aplikační server

Applet se vkládá do HTML stránky pomocí tagu `<applet></applet>`. Tento objekt může obsahovat několik atributů - mezi ty nejhlavnější patří `code` označující hlavní třídu appletu, `width` pro šířku a `height` pro výšku appletu a `archive`, ve kterém jsou jména `.jar` archívů appletu. Mezi další tagy, které ale nejsou povinné patří například `alt`, který zobrazuje text, když nedojde k načtení appletu, případně `codebase`, který odkazuje na složku s třídami a .jar archivy appletu (pokud není použit tag `codebase`, třídy a .jar archivy se musí nacházet ve složce, kde se nachází stránka s appletem).

```
<html>
<head>Applet</head>
<body>
<applet code="Applet_McC.class" width="1000" height="700" archive="app_gui.jar"></applet>
</body>
</html>
```

Výpis 26: Ukázka stránky `index.html` s appletem

Při nasazení servletu na aplikační server záleží na dvou hlavních podmínkách - adresářové struktuře servletu a konfiguračním souboru servletu `web.xml`. U adresáře servletu musíme dvě složky `WEB-INF` a `META-INF`. Do složky `META-INF` se vloží soubor `context.xml`, ve kterém je zapsána kontextová cesta k servletu. Složka `WEB-INF` obsahuje další dvě složky a to `lib` a `classes`. Do adresáře `WEB-INF\lib` se nakopírují všechny `.jar` archivy, které využívá servlet (v našem případě jde o `GUI_final.jar` a `javabuilder.jar`). Adresář `WEB-INF\clasees` obsahuje složku s balíkem servletu a

jeho třídou (u tohoto servletu adresář vypadá takto `WEB-INF\clasees\servlet_package\Servlet_class.class`). Do složky `WEB-INF` se ještě nakopíruje konfigurační soubor servletu `web.xml`.

Soubor `web.xml` slouží k vložení parametrů servletu, hlavně k jeho namapování na aplikačním serveru. Nejprve se v tagu `web-app` nastavuje verze používáného `API servletu` a poté se v tagu `servlet` alokuje třída servletu a pomocí tagu `sevlet-mapping` je nastavena URL adresa k servletu. Jelikož adresáře servletu se v našem případě se nachází přímo ve složce appletu, která se nazývá `applet`, stačí pouze zadat `<url-pattern>/applet</url-pattern>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
    app_2_5.xsd">
    <servlet>
        <servlet-name>applet</servlet-name>
        <servlet-class>servlet_package.Servlet_class</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>applet</servlet-name>
        <url-pattern>/applet</url-pattern>
    </servlet-mapping>
</web-app>
```

Výpis 27: Soubor `web.xml`

6 Návod na práci s GUI

V této časti textu bude popsáno, jak ovládat GUI aplikace pro minimalizaci logických funkcí metodou McCluskey.

Na úvodní obrazovce aplikace se nachází jedno textové pole a dvě skupiny radiobuttonů. Do textového pole se zadává počet vstupních proměnných. Pokud je hodnota chybně zadána, zobrazí se chybové oznámení. Dále se zvolí pomocí radiobuttonů tvar funkce a způsob jejího zadání.

Pokud se zvolí jako způsob zadání `Pravdivostní tabulka`, tak po kliknutí na tlačítko `OK` se zobrazí tabulka, v níž je možné u každého řadku si vybrat, jakou hodnotu bude řádek nabývat (logická 0, logická I nebo X). Kliknutí na tlačítko `Další` se zobrazí upravená tabulka vstupních hodnot.

Byl-li zvolen jako způsob zadání disjunktivní, konjunktivní nebo vektorový tvar, místo tabulky pro zadání hodnot se zobrazí textové pole, do kterého se zadá textový řetězec. Pro konjunktivní nebo disjunktivní tvar tento řetězec může být zadán ve dvou tvarech. Pokud zadáváme úplně zadanou funkci, tak do textového pole zapíšeme hodnoty proměnných, kde nabývá dané logické hodnoty (například: 1,2,3,4,6). Chceme-li zapsat neúplně zadanou funkci, tak proměnné kde neznáme hodnotu logické funkce, zapíšeme do

závorky za známe hodnoty (například: 1,2,3(5,7)). Kliknutím na tlačítko `Další` se zobrazí upravená tabulka vstupních hodnot.

Pro zadání funkce ve vektorvém tvaru musí vstupní řetězec obsahovat výčet všech hodnot vektorů - řetězec může obsahovat hodnoty logická 0, logická I nebo X (například: $f(x_2, x_1, x_0)$=01X1001X).

Po zobrazení upravené tabulky se vytvoří dvě nová tlačítka `Nápověda` a `Doplnit` vedle tlačítka `Další`, které se stane neaktivním. U tabulky se objeví checkboxy, které slouží k označení jednotlivých řádků tabulky, které chceme spojit. Pokud označené řádky splňují podmínku spojení, přidá se nově vzniklý vektor do druhé tabulky. K naplnění druhé tabulky lze využít tlačítka `Doplnit`, které po kliknutí doplní celou tabulku. Tlačítko `Nápověda` zobrazí zprávu s textem, které vektory lze ještě spojit. Pokud je druhá tabulka naplněna, tlačítko `Další` se stane zase aktivním.

Po veškeré minimalizaci se zobrazí výsledná minimalizovaná funkce v textovém poli. Uživatel může poté v nabídce `Menu` - `Nové zadání` spustit nové zadání minimalizace logické funkce, případně zvolením `Menu` - `Ukončit program` vypnout aplikaci.

7 Závěr

V bakalářské práci bylo okrajově popsáno téma logických obvodů a minimalizace logických funkcí metodou McCluskey. Většina textu byla věnována popisu a tvorbě aplikace pro minimalizaci logických funkcí pomocí McCluskeyeho metody a jejií nasazení na webové stránky.

Cílem bylo vytvořit aplikaci v MATLABu, která se umístí na Matlab server Vysoké školy Báňské - Technické univerzity Ostrava. Z důvodů nemožnosti využití naprogramovaného GUI v MATLABu na Matlab serveru, bylo nutné vytvořit Java applet, který se umistí na Matlab server a komunikuje s MATLAB funkcemi pro minimalizaci.

U výsledné aplikace, jak v MATLABu tak v Java appletu, byla upřednostněna funkční stránka a přívětivost ovládaní uživatelem před estetickou líbivostí, protože pro studijní účely není grafická stránka důležitá. U aplikace je možnost jejího dalšího rozšíření například o projení s databází, kde by si uživatelé mohli ukládat už vyřešené logické obvody, další možností by mohlo být vytváření nákresu zapojení logických obvodů s určitými logickými členy pro výslednou minimalizovanou logickou funkci.

Jiří Hrbáček

8 Reference

[1] DIVIŠ, Zdeněk; CHMELÍKOVÁ, Zdeňka; ZDRÁLEK, Jaroslav. *Logické obvody.* 2.vyd. Ostrava: VŠB - Technická univerzita Ostrava, 2008. 152s 978-80-248-1742-8

[2] BERNARD, Jean Michel; HUGON, Jean; Le CORVEC, Robert. *Od logických obvodů k mikroprocesorům.* 2. nezm. vyd. Praha: SNTL - Nakladatelství technické literatury, 1998.686s

[3] ZAPLATÍLEK, Karel; DOŇAR, Bohuslav. *MATLAB - tvorba uživatelských aplikací.* 2. dotisk 1. vyd. Praha: BEN - technická literatura, 2008. 216s 978-80-7300-133-9

[4] Rich Matlab editbox contents. YAIR ALTMAN. [online]. 20.1.2010 [cit. 2012-03-20]. Dostupné z WWW: http://undocumentedmatlab.com/blog/rich-matlab-editbox-contents/

[5] MATLAB Compiler: Product Description. THE MATHWORKS. [online]. © 1994-2012 [cit. 2012-03-20]. Dostupné z: http://www.mathworks.com/products/compiler/description3.html

[6] Web Deployment of MATLAB Applications Guide. THE MATHWORKS. [online]. © 1994-2012 [cit. 2012-03-20]. Dostupné z: http://www.mathworks.com/support/tech-notes/1600/1608.html

[7] Guidelines for Working with MWArray Classes. THE MATHWORKS, Inc. [online]. © 1984-2007 [cit. 2012-03-20]. Dostupné z: http://www.kxcad.net/cae_matlab/toolbox/javabuilder/ug/bqst6t8.html

A Seznam příloh na CD

Adresářová struktura s přílohami na přiloženém CD.

`/prace/zadani_bakalarske_prace.pdf` - zadání bakalářské práce
`/prace/text_bakalarske_prace_hrb109.pdf` - text bakalářské práce
`/matlab/` - složka obsahující m-fily aplikace
`/applet/` - složka obsahující soubory appletu a servletu